DOCUMENT RESUME

ED 195 422 SE 033 583

AUTHOR Gales, Larry
TITLE Design Standards for Instructional Computer Programs. Physical Processes in Terrestrial and Aquatic Ecosystems, Computer Programs and Graphics Capabilities.
INSTITUTION Washington Univ., Seattle. Center for Quantitative Science in Forestry, Fisheries and Wildlife.
SPONS AGENCY National Science Foundation, Washington, D.C.
PUB DATE Sep 78
GRANT NSF-GZ-2980: NSF-SED74-17696
NOTE 24p.: For related documents, see SE 033 581-597.

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Biology: College Science: *Computer Assisted Instruction: Computer Graphics: *Computer Programs: Ecology: Environmental Education: Higher Education: Instructional Materials: *Interdisciplinary Approach: *Physical Sciences: Science Education: Science Instruction

ABSTRACT

These materials were designed to be used by life science students for instruction in the application of physical theory to ecosystem operation. Most modules contain computer programs which are built around a particular application of a physical process. The report describes design standards for the computer programs. They are designed to be uniform in structure and usage, tolerant of student input, easy to use and to operate in either batch or interactive mode and with easy re-direction of input and output. All programs (with a few exceptions) are coded in ANSI Fortran and follow the standards set by the CONDUIT organization. Standard input and output modules are the principle agents for program standardization. The external input is handled by a module called the format free input system which permits a user to assign values to variables by name without regard to position or order. The output is handled by two modules called PRNT3D and PLOT3D. The computational module is invoked by a main program or driver routine which coordinates its activities with those of the input and output modules. All of the programs are guided by the principles of structured programing. (Author/CS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Patricia Babb
of the NSF

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

SE 033 583

DESIGN STANDARDS
FOR INSTRUCTIONAL COMPUTER PROGRAMS

by

Larry Gales

This instructional module is part of a series on Physical Processes in Terrestrial and Aquatic Ecosystems supported by National Science Foundation Training Grant No. GZ-2980

September 1978

DEC 1 2 1980

DESIGN STANDARDS FOR INSTRUCTIONAL COMPUTER PROGRAMS

Introduction

The NSF grant GZ-2980, "Physical Processes in Terrestrial and Aquatic Ecosystems," is concerned with the development of instructional units, called modules, which teach aspects of physical processes to students in the life sciences. Most modules contain computer programs which are built around a particular application of a physical process. The program design standards are aimed at enhancing the usage, portability, ease of modification, and dissemination of these programs. In particular, the programs are designed to be uniform in structure and usage, tolerant of student input, easy to use, and to operate in either batch or interactive mode and with easy re-direction of input and output. All programs are coded in ANSI Fortran (with a few carefully isolated and well documented exceptions permitted) and closely follow the standards set by the CONDUIT organization.

Standard input and output modules are the principal agents for program standardization. All external input is handled by a module called the format free input system (Anderson and Gales, 1978) which permits a user to assign values to variables by name without regard to position or order. This system is easy to use, permits self-documenting input, offers helpful diagnostics (unlike most other input systems such as Fortran's non-standard NAMELIST), is suitable for all but very large data sets, and features a default value structure which permits a student to exercise a graduated degree of control over the program which expands as his knowledge and skill increase.

The output is handled by two modules called PRNT3D and PLOT3D (Gales 1978a, 1978b). The former is a very flexible and portable printer plot

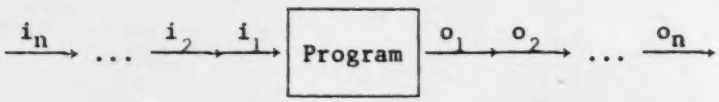
package which features point, multi-line, and simulated three-dimensional surface plots, and is suitable for the display of many physical processes, while the latter is a less portable three-dimensional hidden line Calcomp-type display. Both have similar interfaces to the calling program.

With the input and output modules fixed, the only variable left is the computational module, and this is invoked by a main program, or driver routine, which coordinates its activities with those of the input and output modules.

Another agent of standardization is the approach offered by structured programming (SP) which emphasizes simplified control paths, single purpose routines developed by step wise refinement, and single entry - single exit units with well defined interfaces. All of the programs are guided by the principles of SP and some of them (including the three support modules) are coded in a literal structured Fortran which adheres strictly to the half-dozen control structures of SP (Gales, 1975 and Appendix 3).

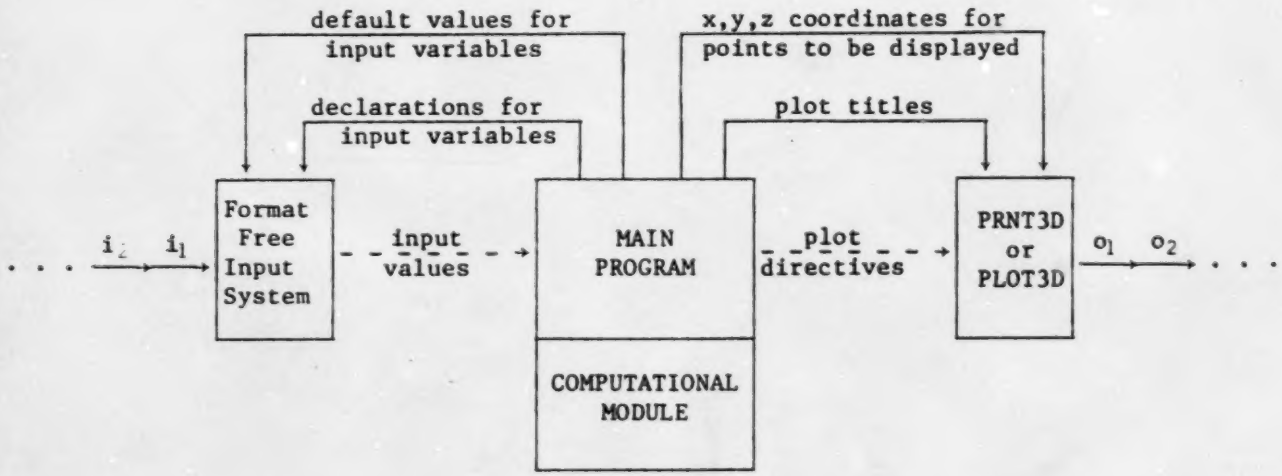
Logical Structure of Programs

To the user, each program appears as follows:



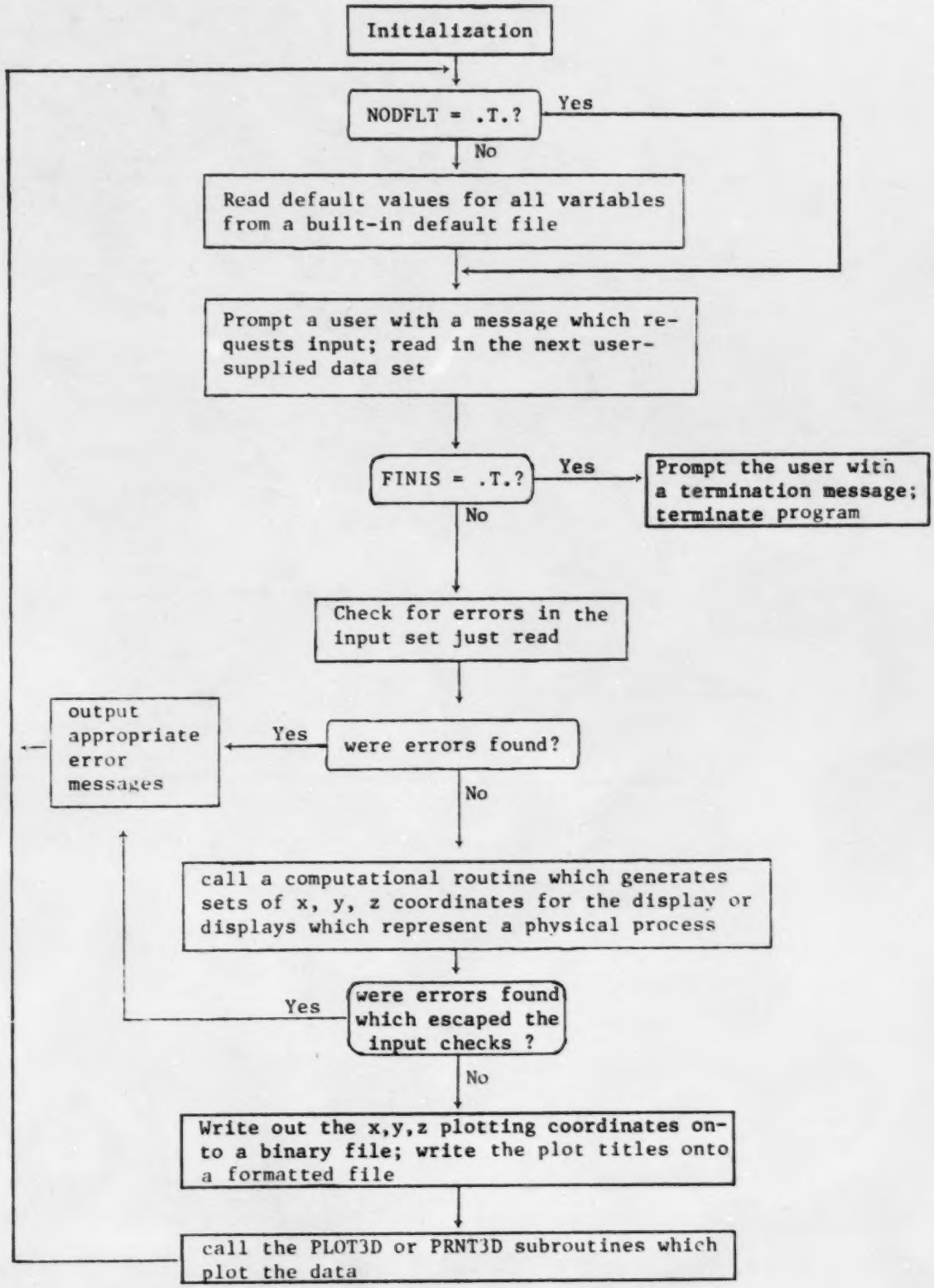
where i_j is the jth input data set and o_j is its associated set of PRNT3D or PLOT3D displays.

To a programmer, each program looks like this:



where the solid lines indicate passage of information by way of files, and the dotted lines indicate communication via common blocks. All files are formatted except for the file of x,y,z coordinates which is in binary.

The main program, which coordinates the activities of the input, output, and computational modules, consists of an initialization section, a read-process-output loop, and a termination section, and is structured as follows:


Initialization
NODFLT = .T.?
Yes
No
Read default values for all variables from a built-in default file
Prompt a user with a message which requests input; read in the next user-supplied data set
FINIS = .T.?
Yes
Prompt the user with a termination message; terminate program
No
Check for errors in the input set just read
were errors found?
Yes
output appropriate error messages
No
call a computational routine which generates sets of x, y, z coordinates for the display or displays which represent a physical process
were errors found which escaped the input checks ?
Yes
No
Write out the x,y,z plotting coordinates onto a binary file; write the plot titles onto a formatted file
call the PLOT3D or PRNT3D subroutines which plot the data

The variables NODFLT and FINIS are standard input variables which are used to suppress the input of default values, and to terminate every program, respectively. The main program always calls six special purpose routines:

NMLIST - a small routine which writes a formatted file containing the names, dimensions, and types of all input variables.

QQREAD - a routine in the format free input system which handles input.

WRTDFF - a small routine which writes out default values for all input variables on a formatted file to be read by QQREAD.

WRTTLF - a small routine which writes out a set of titles for PRNT3D or PLOT3D on a formatted file.

WRTXYF - a small routine which writes x,y,z coordinates onto a binary file for use by PRNT3D or PLOT3D.

QQPR3D - entry points in PRNT3D or PLOT3D.
or
QQPL3D

Physical Structure of Programs

The physical structure of a program, i.e., the choice of identifiers, size and format of routines, file structures and formats, subroutine linkages, etc., should reflect its logical organization.

Identifiers are 1-6 character names assigned to values, common blocks, programs, subroutines, functions, and files. Identifiers should

- be as few as possible
- have a single meaning which remains invariant throughout a routine
- correspond to some reality outside the program; i.e., should represent a part, process, or concept associated with the problem, not the program
- possess mnemonic significance

The choice of indentifier names is important because they govern the readability of code more than any other single factor. A well written program has very few identifiers with changing or multiple meanings; their presense in large numbers is a sure sign of a poorly structured and overpatched program with many unstated limitations and undetected bugs.

Each routine should consist of a header and a body. The header contains declarations and comments and primarily defines what is being operated on. The body consists of executable code and defines the operations to be applied to the data described in the header. The body should be fairly small (about one page), should perform a single basic purpose, should be entered at the top and exited at the bottom, and should have relatively few, but well defined, connections to other routines.

The header provides the main internal documentation for the program, serves as a checklist against which code can be compared, and discourages poorly written code. One cannot easily graft a good header onto bad code, because the many variables with their changing meanings and non-mnemonic spellings cannot be declared and defined in the header in a compact and understandable way. It is essential that the header and body evolve together, with the header serving as a censor of bad design. Since the header reflects the code, cumbersome and difficult headers arise only from poorly written code.

The header for the main program consists of a number of sections each of which is delimited by vertical blank spaces and a line of dashes. The sections are as follows:

Purpose - a brief description of the purpose of the program including the main execution steps in order of occurrence.

Identification - an identification number, date, location, language, computer installation, and programmer name(s).

Global variables - a group of common blocks each structured as follows: descriptive comments enclosed in a star box, the common block, explicit type declarations for all of its variables, and comment cards which define all variables.

Local variables - explicit type declarations and definitions for all local variables.

Functions - explicit type declarations and definitions for all but standard ANSI intrinsic and external functions.

Subroutines - names of all subroutines called by this routine.

Externals - the names, purposes, and algorithms for all subroutines and functions which are unavailable or are coded in machine-dependent language.

Files - the names, types (formatted or binary), direction (input, output, input/output), and data items for all files used in the program.

Restrictions - a list of limitations.

Error Handling - a list of error conditions and actions.

Non-ANSI Usage - a list of non-ANSI usages.

Machine Dependencies - a list of machine dependent features, e.g. word length dependencies.

Constants - a set of DATA statements which assign values to all constants in the routine.

Initialization - a set of assignments and loops which explicitly initialize all variables which require starting values.

Start - the start of the body of the routine.

A complete form for the header is presented in Appendix I.

Subroutines and functions contain linkages, headers, and bodies. Subroutine linkages are structured as follows. The arguments are separated into three classes: input, input/output, and output arguments. The input arguments are listed first, the input/output arguments are listed next and are separated from the input arguments by several spaces, and the output arguments are listed last, and are separated from the input or input/output list by several spaces. For example,

```
SUBROUTINE   SUB1 (X,Y,      PX,      A,BOT)
                    input    input/   output
                             output

SUBROUTINE   SUB2 (          P,QQ,T           )
                             input/
                             output

SUBROUTINE   SUB3 (X,TOT,                B,ZR)
                    input                output
```

Such a convention clearly reveals the nature of the linkages. The convention also applies to those statements which call the subroutines, for example, CALL SUB3 (X,TOT, B,ZR). All the arguments for functions should be input arguments only.

The headers for subroutines and functions differ slightly from the main program header, in that they:

- have an "Arguments" section
- lack an "Identification" section (unless it differs from the main program)
- lack discriptive comments and definitions for common block variables; and unused variables in the common block are declared with a ZZZZn (*) convention
- lack a "Files" section

End-of-file Handling

The end-of-file marker, whose full treatment is undefined in ANSI Fortran, is needed to delimit input data sets read by the format free input system, and to terminate files of x,y,z coordinates. In the former case, the end-of-file is represented by a dollar sign ($), and in the latter case by the x,y,z triple (-99999., -99999., -99999.)

Error Handling

In most cases, errors which arise or are detected within a subroutine are handled as follows:

1) the error condition triggers a message of the form:

******ERROR NO. n DETECTED IN name

<error message>

<the names and values of variables which closely relate to the error>

where n is the error number and name is the name of the subroutine. The error message is written out on a special error message file, although the latter may be equated to the standard output file.

2) the error number is passed back to the calling routine via common or an argument list.

3) the subroutine is aborted and control returns (through various subroutine levels if necessary) to the main program.

4) the main program zeros out all error indicators and skips to the next input data set.

The above procedure permits the processing and error scanning of multiple sets of input when the program is run in batch mode, and insures easy recovery from errors by the user when the program is run interactively.

Input

All external input is handled by the format free input system which automatically checks for name, value, and subscript violations. Each program also contains a special subroutine, named INCHK, which tests each input variable v against a range, vMIN to vMAX, or set of permissable values. Values outside this range trigger error messages. The vMIN and vMAX values are set by data statements within INCHK.

Output

All output is designed to fit on standard 8½ by 11 inch pages for easy storage and dissemenation. The output consists of prompter messages, echoed input, error messages, and the plotted output from PRNT3D or PLOT3D along with their associated titles and other annotation.

External Documentation

Each instructional module is presented as a package which is described by the following packing list:

IMPORTANT

Packing List for: program <name>

Enclosed with this list are the following elements comprising the package:

1 each -- Magnetic tape containing a) the source program plus support routines, and b) sample runs with annotated control cards and input cards.

1 each -- CONDUIT Tape Distribution Form describing how the tape was recorded and what it contains. This form contains essential information and should be carefully read.

1 each -- Listing of the tape as read by us before we sent it.

1 each -- Listing of the source program plus support routines as compiled under the Minnesota Fortran ANSI verifier compiler.

1 each -- Module description plus problem sets, program abstract, and user's guide.

1 each -- Design Standards for Programs.

1 each -- Program abstract, user's guides, and programmer's guides for all support routines.

1 each -- Notes for implementers.

The user's guide consists of the following sections:

1) Identification - contains name, author, date, and location
2) Purpose
3) Operation
4) Program organization - an aggregate flow chart of the program
5) Input - a table of all input variables with their names, types, dimensions, range limits, purpose, and default values
6) Output - a verbal discussion of the output produced by the program
7) Restrictions
8) Error messages
9) Sample runs - a complete input set including annotated control cards and an annotated set of input data sets, followed by the exact outputs which result. The sample run should permit users at a different installation to test the program.
10) References

The programmer's guide for each of the Physical Processes computer modules consists of a note to implementers, internal comments in the computer program, and this document on design standards. The three support routines FFORM, PRNT3D, and PLOT3D, however, which handle input/output for each module, have detailed programmer's guides as separate documents.

Testing Procedures

Testing procedures should insure that all subscripts remain within array bounds, all control paths are fully exercised, all boundary conditions are tested, and all procedures are numerically stable.

APPENDIX I: FORMAT OF PROGRAM HEADERS

The header should be formatted as follows:

```
card columns
1    6 7              21       29     37        45       53       61

C
C
C-PURPOSE-------------------------------------------------------------
C
C       comments describing the purpose and main execution steps
C
.
.
.
C
C
C-IDENTIFICATION------------------------------------------------------
C
C       I.D. number, date, location, language, computer installation,
C       programmer name(s)
.
.
.
C
C
C-GLOBAL VARIABLES----------------------------------------------------
C
C       ********************************************************
C       *  purpose and description of following common block   *
C       *                                                      *
C       ********************************************************
C
      .  COMMON/name/  v,      v,     v,       v,      v,      v,
      .                v,      v,     v,       . . .
      .                .
      .                .
      .                .
         INTEGER       v,      v,     v, . . .
         REAL          v,      v,     v, . . .
         LOGICAL       v,      v,     v, . . .
C
C       DEFINITIONS
C
C       v=             verbal description of variable
C       v=
         .
         .
         .
C
C
C-LOCAL VARIABLES ----------------------------------------------------
C        INTEGER       v,      v,     v, . . .
         REAL          v,      v,     v, . . .
         LOGICAL       v,      v,     v, . . .
C
```

APPENDIX I continued

```
1   6 7             21      29      37      45      53      61

C       DEFINITIONS
C
C       v=          verbal description of variable
C       v=
        .
        .
        .
C
C
C-FUNCTIONS-----------------------------------------------------------
C
        INTEGER     fcn,    fcn,    fcn,    . . .
        REAL        fcn,    fcn,    fcn,    . . .
        LOGICAL     fcn,    fcn,    fcn,    . . .
C
C       DEFINITIONS
C
C       fcn=        verbal description of purpose of function
C       fcn=
        .
        .
        .
C
C
C-SUBROUTINES---------------------------------------------------------
C                   sub,    sub,    sub,    . . .
C                   .
                    .
                    .
C
C
C-EXTERNALS-----------------------------------------------------------
C
C       sub         purpose, algorithm, and interface to rest
C                   of program for each subroutine or function
C                   which is highly machine dependent or is not
C                   included in the source deck
        .
        .
        .
C
C-FILES---------------------------------------------------------------
C
C       f           the type (formatted or binary), direction
C                   (in, out, or in/out), and data items
C                   for file f
        .
        .
        .
```

APPENDIX I continued

```
1    6 7              21       29      37       45       53       61

C
C
C-RESTRICTIONS-------------------------------------------------------
C
C       A discussion of program limitations
        .
        .
        .
C
C
C-ERROR HANDLING-----------------------------------------------------
C
C       A list of error conditions and actions
        .
        .
        .
C
C
C-NON-ANSI USAGES----------------------------------------------------
C
C       List of non-ANSI features
        .
        .
        .
C
C
C-MACHINE DEPENDENCIES-----------------------------------------------
C
C       Discussion of the effects of such things as word lengths,
C       numerical precision, and memory requirements
        .
        .
        .
C
C
C-CONSTANTS----------------------------------------------------------
C
        DATA          v, v, ..., v   /       . . .   /
C
C
C-INITIALIZATION-----------------------------------------------------
C
        All variables which require starting values are
        initialized here.  Also files may be rewound
        .
        .
        .
C
C
C-START--------------------------------------------------------------
C
        The body of the program starts here
```

APPENDIX I continued

Subroutines and functions differ from the main program in that they have an arguments section and generally lack identification and files sections, often do not comment the common blocks, and specify unused common variables with the ZZZZn(*) convention. The first part of a subroutine header is as follows:

```
1   6 7              21      29      37      45      53      61
      SUBROUTINE name(a,a,    a,a,    a,a)
C
C
C-PURPOSE-------------------------------------------------------------
      .
      .
      .
C
C
C-ARGUMENTS-----------------------------------------------------------
C
      INTEGER        a,      a,      a,      . . .
      REAL           a,      a,      a,      . . .
      LOGICAL        a,      a,      a,      . . .
```

APPENDIX II: ANSI FUNCTIONS

Allowable Intrinsic Functions:

ABS	IABS	DABS		
AINT	INT	IDINT		
AMOD	MOD			
AMAXO	AMAXI	MAXO	MAXI	DMAXI
AMINO	AMINI	MINO	MINI	DMINI
FLOAT	IFIX			
SIGN	ISIGN	DSIGN		
DIM	IDIM			
SNGL	REAL			
AIMAG	DBLE	CMPLX	CONTG	

External functions:

EXP	DEXP	CEXP	
ALOG	DLOG	CLOG	
ALOG10	DLOG10		
SIN	DSIN	CSIN	
COS	DCOS	CCOS	
TANH	SQRT	DSQRT	CSQRT
ATAN	DATAN	ATANH	DATANH
DMOD	CABS		

APPENDIX III: STRUCTURED FORTRAN

Structured Fortran uses the familiar control structures of SP:

1. IF b THEN s END-IF
2. IF b THEN s_1 ELSE s_2 END-ELSE
3. IF b_1 THEN s_1 ELSEIF b_2 THEN s_2 ... OTHERWISE s_n END-ELSEIF
4. WHILE b DO s END-WHILE
5. REPEAT s UNTIL b END-REPEAT
6. LOOP s WHILE b DO s END-LOOP
7. DO s CONTINUE (a restricted repeat statement)

where b, b_1, ... denote Boolean expressions and s, s_1, ... denote any set of statements including other control structures. The above structures are translated into left and right hand sides in Fortran as described below. The translation introduces one non-ANSI feature which can easily be removed by a very small preprocessor. For a discussion of the use of this technique see Gales, 1975. The control structures are as follows:

II. Structure

```
1    67              21      25
```

```
                     IF
     .(.NOT.(              b
     .))GOTO101;       THEN
                           s
   101 CONTINUE;     END-IF
```

```
                     IF
     .(.NOT.(              b
     .))GOTO201;       THEN
                           s1
      GOTO202;         ELSE
   201                     s2
   202 CONTINUE;     END-ELSE
```

```
                     IF
     .(.NOT.(              b1
     .))GOTO301;       THEN
                           s1
      GOTO307;         ELSEIF
   301 IF (.NOT.(           b2
     .))GOTO302        THEN
                           s2
                          .
                          .
                          .
      GOTO307;         ELSEIF
   305 IF (.NOT.(           b6
     .))GOTO306;       THEN
                           s6
      GOTO307;         OTHERWISE
   306                     s7
   307 CONTINUE;     END-ELSEIF
```

```
C                    WHILE
   401 IF (.NOT.(          b
     .))GOTO402;       DO
                           s
      GOTO401;       END-WHILE
   402 CONTINUE
```

```
                     DO   701  I=1,N
                           s
   701               CONTINUE
```

II. Structure(continued)

```
1     7              21       25
-------------------------------------------------------
C                    REPEAT
  501                       s
C                        UNTIL
      IF (.NOT.(            b
     .))GOTO501;     END-REPEAT
-------------------------------------------------------
C                    LOOP
  601                       s1
C                        WHILE
      IF (.NOT.(            b
     .))GOTO602;        DO
                            s2
      GOTO601;       END-LOOP
  602 CONTINUE
-------------------------------------------------------
```

References

Anderson, L. and L.E. Gales. 1978. Programmer's guide for subroutine FFORM: a format free input system. Center for Quantitative Science in Forestry, Fisheries, and Wildlife, University of Washington, Seattle, Washington.

Gales, L.E. 1975. Structured Fortran with No Preprocessor. SIGPLAN Notices, October.

Gales, L. 1978. Programmer's guide for subroutine PRNT3D. Center for Quantitative Science in Forestry, Fisheries, and Wildlife, University of Washington, Seattle, Washington.

Gales, L. 1978. Programmer's guide for subroutine PLOT3D. Center for Quantitative Science in Forestry, Fisheries, and Wildlife, University of Washington, Seattle, Washington.